并查集

2019年3月8日 15:08

-
- 并查集

1. 并查集disjoint-set：动态维护若干不重叠的集合的数据结构，支持查询和合并
   1) 查询get：查询该元素属于哪个集合
   2) 合并merge：将两个集合合并成一个大集合
2. 代表元法：每个集合选一个固定的元素作为代表
   1) 集合中每个元素都直接指向代表，查询极快，但合并极慢
   2) 集合中每个元素间接指向树根，查询可能稍慢，但合并极快
3. 树根代表元法的提效方法
   1) 路径压缩：执行get操作时让路径上每个元素都直接指向树根，每次get操作的均摊复杂度为O（logN）
   2) 秩：树的深度或集合的大小，一般要设法记录在代表元
   3) 按秩合并：合并时让小秩树根作为大秩树根的根节点，每次get操作的均摊复杂度为O（logN）
   4) 启发式合并：并查集上体现为设集合大小为秩的按秩合并
   5) 同时使用路径压缩和按秩合并的get操作均摊复杂度可再降为O（α(N)），α值反阿克曼函数，比log长得还慢
4. 只有路径压缩的并查集实现：
   1) 初始化：

```
for__(i,1,n){
    fa[i]=i;
}//初始化为n棵平凡树
```

   2) 查询：

```
int get(int x){
    if(fa[x]==x)
        return x;          //递归终点
    else
        return fa[x]=get(fa[x]);  //路径压缩
}
```

   3) 合并：

```
void merge(int x,int y){
    if(get(x)!=get(y))
        fa[get(x)]=fa[y]; //默认x变成y的子树
}
```

```
int rt[MN];
int fd(int x){ return rt[x]==x? x: rt[x]= fd(rt[x]); }
void mg(int x, int y) {if(fd(x)!=fd(y)) rt[rt[y]]=rt[x];}
```

5. 边带权的并查集实现（实现记录链形并查集点到根距离，每次合并是并到最末尾）
   1) 初始化：注意集合大小初值是1，但距离初值是0

```
for_(i,1,MN)
```

```
        rt[i]=i,
        sz[i]=1,        //集合的大小，初值1
        dst[i]=0;       //到根的距离，初值0
```

2) 查询：父亲需要更新，说明旧的父亲并到了新的父亲后，此时旧父亲到根的距离就是当前点到新父亲的新增距离

```
int fd(int x){
    if(rt[x]==x)
        return x;
    int r=fd(rt[x]);            //先存下可能未更新好的新根
    dst[x]+=dst[rt[x]];         //+=老的根的新的长度
    return rt[x]=r;             //再用新的根路径压缩
}
```

3) 合并：为了把到根距离更新给子结点，需要更新合并到链后的新dst，为了给孙结点更新dst，紧接着还要把sz更新

```
inline bool need(int a,int b){
    return fd(a)!=fd(b);
}       //一定要保证先在need压缩好路径，再使用下面的merge!
inline void mg(int a,int b){        //a插到b的尾部
    int ra=rt[a];
    int rb=rt[b];
    rt[ra]=rb;                  //合并
    dst[ra]=sz[rb];  //先更新a根前面的点数
    sz[rb]+=sz[ra];  //再更新b的新大小
}
```

-
- 例题

1. 同性警告：n人m组，每组两异性，如果发现存在同性小组，输出警告
   1) 起：异性放在不同并查集，如果发现小组中两个人在同一并查集就输出警告
   2) 转：不知道谁是什么性别，只知道我和队友的性转是同性，可以给根数组开两倍，给下标+n的根数组存储我的性转的根，这种做法称为扩展域

```
scanf("%d%d",&n,&m);
for__(i,1,n){
    rt[i]= i;
    rt[i+n]= i+n;}
bool wn= 0;
while(m--){
    scanf("%d%d",&a,&b);
    if(wn)   //已经输出warning就不处理接下来的输入了
        continue;
    merge(a,b+n);
    merge(b,a+n);
    if(find(a)==find(b))//带路径压缩的话，这里find()可以换成rt[]
        wn=1;}
if(wn)
    printf("Warning\n");
else
    printf("Ok\n");
```

2. 舰队规模查询：操作1合并两船所在舰队，操作2摧毁船，操作3循环所在舰队船数
    1) 需要开一个数组存储每个秩（容量）只有根的容量是有意义的
    2) 需要开另一个数组存储船有没有被摧毁
    3)
```
const int MN = 50005;
int rt[MN], rk[MN];
bool wrecked[MN];
int find(int x){
    if(x==rt[x])
        return x;
    else
        return rt[x]= find(rt[x]);}
void merge(int x, int y){ //默认x并给y
    if(find(x) == find(y))      //避免重复合并
        return ;
    rk[rt[y]]+= rk[rt[x]];
    rt[rt[x]]= rt[y];}
```
    4)
```
for__(i,1,n)
    rt[i]= i,
    rk[i]= 1;
int op,x,y;
for_(i,0,m){
    cin>>op>>x;
    switch(op){
        case 1:
            cin>>y;
            merge(x,y);
            break;
        case 2:
            if(!wrecked[x])             //避免重复击落
                wrecked[x]= 1,
                --rk[find(x)];
            break;
        default:
            cout<<rk[find(x)]<<endl;}}
```
3. 银河英雄传说
    1) 套边带权并查集即可，链上距离即为到根距离差-1
    2)
```
scanf("%d",&t);
while(t--){
    scanf(" %c%d%d",&c,&a,&b);
    if(c=='M'){
        if(need(a,b))
            mg(a,b);
    }else{   //询问
        if(need(a,b))
            printf("-1\n");
        else
            printf("%d\n",abs(dst[a]-dst[b])-1);}}
```
4. 摧毁部分船后的舰队数量
    1) 在线做很难，但是存下摧毁顺序，离线反向存答案就简单了
    2) 把摧毁的船视作不存在，反着依次把每个船建回去，求对连通分支数的贡献
    3)
```
const int MN= 200005;
int n,m,k;
```

```
int x,y;
vector<int> tmn[MN];
int t[MN],ans[MN];      //摧毁顺序，输出答案
int rt[MN],cnt;
bool brk[MN];

int find(int x){
    if(rt[x]==x) return x;
    return rt[x] = find(rt[x]);}
inline bool need_merge(int x, int y){
    return find(x)!=find(y);}
inline void merge(int x, int y){
    rt[rt[x]] = rt[y];}
```

4)
```
cin>>n>>m;
for__(i,1,n)
    rt[i] = i;
for__(i,1,m)
    cin>>x>>y,
    tmn[x].push_back(y),
    tmn[y].push_back(x);
cin>>k;
for__(i,1,k){
    cin>>t[i];
    brk[t[i]] = 1;}
for__(x,1,n)
    if(!brk[x])
        for(auto y : tmn[x])
            if(!brk[y])
                merge(x,y);
for__(i,1,n)
    if(!brk[i] && rt[i]==i)
        ++cnt;
ans[k]= cnt;
rof_(i,k,1){
    int x= t[i];
    brk[x]= 0;
    ++cnt;
    for(auto y : tmn[x])
        if(!brk[y] && need_merge(x,y))
            --cnt,
            merge(x,y);
    ans[i-1] = cnt;}
for__(i,1,k)
    cout<<ans[i]<<'\n';
```

5. 排座位：1代表是朋友，-1代表是死对头，朋友关系是对称、可传递的，死对头关系是对称，不一定传递的，求任两个人的关系

1)
```
int fa[105];      //并查集存储朋友关系
bool noway[105][105]; //人少，偷个懒用二维数组存敌对关系
int get(int x){    //求集合代表元素
    if(fa[x]==x)
        return x;
    else
        return fa[x]=get(fa[x]);}
void merge(int x,int y){
```

```
    fa[get(x)]=get(y);}
int main(){
    int n,m,k,l,r,t;
    sd(n);  sd(m);  sd(k);
    for__(i,1,n)
        fa[i]=i;//初始化为n棵平凡树
    while(m--){
        sd(l);   sd(r);   sd(t);
        if(t==1) //是朋友
            merge(l,r);
        else            //是敌人
            noway[l][r]=noway[r][l]=1; }
    while(k--){
        sd(l);   sd(r);
        //注意合并时部分结点没及时更新，所以要get代替fa[]
        if(get(l)==get(r) && !noway[l][r])
            puts("No problem");
        else if(get(l)!=get(r) && !noway[l][r])
            puts("OK");
        else if(get(l)==get(r) && noway[l][r])
            puts("OK but...");
        else if(get(l)!=get(r) && noway[l][r])
            puts("No way");}
```

6. 房产排名

1) 亲人关系是传递的，题目要输出最小序号，合并时可以考虑序号

```
struct Ans{
    int id,n; //编号，人口
    double t,m;     //人均套数，人均面积
    bool operator<(const Ans& a)const{
        if(fabs(m-a.m)>1e-5)
            return m>a.m;
        else
            return id<a.id;}
}ans[10005];

int r[10005],n[10005],t[10005],m[10005],v[10005],cnt;

int root(int i){
    if(r[i]==i)
        return i;
    else
        return r[i]=root(r[i]);}

void merge(int n,int N){
    int a=root(n);
    int A=root(N);
    if(a<A)  //让大根指向小根
        r[A]=a;
    else
        r[a]=A;}

int main(){
    int N,id,pp,mm,k,kid;
```

```
scanf("%d",&N);
for_(i,0,10000)
        r[i]=i;                 //并查集初始化!
for_(i,0,N){
        scanf("%d%d%d%d",&id,&pp,&mm,&k);
        v[id]=1;
        if(pp!=-1)
                merge(id,pp),
                v[pp]=1;
        if(mm!=-1)
                merge(id,mm),
                v[mm]=1;
        for_(j,0,k){
                scanf("%d",&kid);
                merge(kid,id);
                v[kid]=1;}
        scanf("%d%d",t+id,m+id);}
for_(i,0,10000)
        if(v[i] && root(i)!=i)
                n[r[i]]++,
                t[r[i]]+=t[i],
                m[r[i]]+=m[i];
for_(i,0,10000)
        if(v[i] && r[i]==i)
                ++n[i],
                ans[cnt].id=i,
                ans[cnt].n=n[i],
                ans[cnt].t=(double)t[i]/n[i],
                ans[cnt].m=(double)m[i]/n[i],
                ++cnt;
cout<<cnt<<endl;
sort(ans,ans+cnt);
for_(i,0,cnt)
        printf("%04d %d %.3f %.3f\n",ans[i].id
                ,ans[i].n,ans[i].t,ans[i].m);
return 0;}
```

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix. --------我是底线----------